Script di connessione

di Gianni Tomasicchio

Le operazioni di connessione e selezione del database sono così comuni all'interno degli script PHP che spesso si preferisce inserirle in un file indipendente, da richiamare negli script che ne hanno bisogno. Il codice è sempre lo stesso, pertanto conviene scriverlo una volta sola, salvarlo in un file ed includerlo dove necessario.

connect.php

```
<?php
$link = mysql_connect('localhost', 'nome_utente', 'password');
if (!$link) {
        die ('Non riesco a connettermi: ' . mysql_error());
}

$db_selected = mysql_select_db('prova', $link);
if (!$db_selected) {
        die ("Errore nella selezione del database: " . mysql_error());
}
?>
```

Ecco come potrebbe presentarsi il generico script che fa uso del precedente codice di connessione:

```
<?php
// richiamo lo script responsabile della connessione a MySQL
require 'connect.php';

// il resto dello script: posso effettuare query
// senza preoccuparmi della connessione a MySQL
?>
```

Si noti però che il riutilizzo dello scipt connect.php in altri contesti (altri programmi PHP/MySQL) non è ancora immediato, poiché al suo interno sono presenti diversi dati generalmente soggetti a variazione (host, username, password e nome del DB). Per questo motivo risulta comodo separare tali informazioni dal codice ed inserirle in un apposito file di configurazione config.php, come mostrato di seguito:

config.php

```
<?php
$DB_host     = 'localhost';
$DB_user     = 'nome_utente';
$DB_password = 'password';
$DB_name     = 'prova';
?>
```

connect.php

```
<?php
$link = mysql_connect($DB_host, $DB_user, $DB_password);
if (!$link) {
        die ('Non riesco a connettermi: ' . mysql_error());
}
```

```
$db_selected = mysql_select_db($DB_name, $link);
if (!$db_selected) {
        die ("Errore nella selezione del database: " . mysql_error());
}
?>
```

Il generico script pertanto diventa:

```
<?php
// richiamo il file di configurazione
require 'config.php';

// richiamo lo script responsabile della connessione a MySQL
require 'connect.php';

// il resto dello script: posso effettuare query
// senza preoccuparmi della connessione a MySQL
?>
```

Al di là degli esempi mostrati in questa lezione ciò che conta è rendersi conto che bisogna fin da subito organizzare lo script affinché risulti snello, di facile manutenzione e, almeno in parte, riutilizzabile. Come vedremo, le applicazioni PHP/MySQL tendono con estrema facilità a "complicarsi", sopratutto quando si deve far interagire l'utente con i dati presenti nel database. Gli script che realizzeremo nelle prossime lezioni faranno uso dei file config.php e connect.php qui presentati.

Inserimento dei dati

di Gianni Tomasicchio

Adesso che sappiamo connetterci al server MySQL vediamo come effettuare le query. Cominciamo con un inserimento dati, che segue la sintassi:

```
INSERT INTO
   nome_tabella
SET
   nome_campo_1 = valore_1,
   nome_campo_2 = valore_2
```

oppure:

```
INSERT INTO nome_tabella
   (nome_campo_1, nome_campo_2)
VALUES
   (valore_1, valore_2)
```

In generale per lanciare una query in uno script PHP dobbiamo semplicemente chiamare la funzione **mysql_query()** a cui passare come parametro la stringa contenete la query. Nel caso di un inserimento mysql_query() restituisce TRUE o FALSE, a seconda dell'esito dell'operazione. Ci basta quindi controllare tale valore per verificare se i dati sono stati correttamente immessi nella tabella. In caso negativo con la funzione **mysql_error()** possiamo ottenere maggiori indicazioni su cosa è andato storto.

Il seguente esempio fa riferimento al database ed alla tabella presentati nella Lezione 2. La connessione avviene grazie agli script config.php e connect.php presentati nella Lezione 4.

```
<?php
// richiamo il file di configurazione
require 'config.php';

// richiamo lo script responsabile della connessione a MySQL
require 'connect.php';

// preparo la query
$query = "INSERT INTO utenti (nome,email,sesso,newsletter,attivita,messaggio)
                VALUES ('Mario Rossi','m.rossi@email.it',1,1,2,'inviatemi del
materiale illustrativo')";

// lancio la query
$result = mysql_query($query);

// controllo l'esito
if (!$result) {
        die("Errore nella query $query: " . mysql_error());
}

// chiudo la connessione a MySQL
mysql_close();

echo 'Query eseguita correttamente';
?>
```

E' importante osservare che si è preferito non passare direttamente alla funzione mysql_query() la stringa contenente la query. Tale stringa è stata memorizzata nella variabile $query perché potrebbe risultare utile mostrarla in caso di errore, soprattutto se è stata costruita dinamicamente in funzione di dati provenienti dall'utente (caso mostrato più avanti).

Se avessimo effettuato due o più connessioni a diversi server MySQL, per specificare a quale di questi stiamo inviando la query avremmo dovuto passare alla funzione mysql_query() come secondo parametro anche l'identificativo di connessione ottenuto durante la procedura di connessione al server.

Vediamo ora come eseguire la query di inserimento nel caso in cui i dati provengano da un form compilato da un utente. L'interazione con l'utente complica l'esecuzione della query poiché questa volta non sappiamo a priori cosa verrà inviato con il form. Sono necessarie diverse precauzioni quindi e nel seguente esempio ci concentreremo su quelle che assicurano la creazione di una query corretta, tralasciando altri controlli di validità dei dati non meno importanti ma al di la degli scopi di questo corso.

Lo script che segue si occupa di presentare all'utente un form composto da diversi campi (nome, email, sesso, newsletter, attivita, messaggio) appartenenti a differenti tipologie di controlli (textbox, radio button, checkbox, lista, textarea). Quando l'utente invia il form, lo script estrae dall'array $_POST le informazioni necessarie a costruire la query di inserimento.

Uno dei principali problemi di questo processo è costituito dall'inserimento all'interno della query di stringhe arbitrarie inviate dall'utente. Sappiamo infatti che in una query le stringhe vanno racchiuse da apici:

```
INSERT INTO elenco_film SET titolo = 'Al bar dello sport'
```

ma cosa succede se la stringa contiene a sua volta un apice (apostrofo)?

```
INSERT INTO elenco_film SET titolo = 'L'allenatore nel pallone'
```

E' evidente che questa query non è corretta, infatti non è possibile distinguere l'apostrofo del titolo del film dall'apice che delimita la fine della stringa. Per evitare l'ambiguità la query va corretta nel seguente modo, inserendo il carattere \ (backslash) prima dell'apostrofo:

```
INSERT INTO elenco_film SET titolo = 'L\'allenatore nel pallone'
```

Il PHP possiede una funzionalità chiamata **magic quotes GPC** che consiste nell'eseguire automaticamente l'**escape**, ovvero l'inserimento di un backslash, appena prima di un ' (apice singolo), " (apice doppio), \ (backslash) e NUL per tutti i dati provenienti da GPC (Get, Post e Cookie), ma solo se nel php.ini è abilitata la direttiva magic_quotes_gpc. Sembrerebbe quindi che il problema sia immediatamente risolvibile abilitando tale direttiva.

In realtà la questione è più complessa, infatti ci sono altri caratteri che potrebbero creare dei problemi se inseriti in una query ed inoltre questi dipendono anche dal set di caratteri impiegato nella connessione a MySQL. L'approccio più corretto quindi è disabilitare la funzionalità "magic quotes GPC" ed affidarsi ad una particolare funzione, **mysql_real_escape_string()**, che effettua il corretto escape di tutti i caratteri potenzialmente pericolosi, tenendo in conto il particolare set di caratteri impiegato nella connessione tra PHP e MySQL (alcuni programmatori utilizzano la funzione addslash() al posto di mysql_real_escape_string(), approccio non del tutto errato ma sicuramente meno specifico e sicuro per un database MySQL).

Se non è possibile disabilitare il "magic quotes GPC", oppure se vogliamo realizzare uno script indipendente da tale funzionalità, è possibile procedere in questo modo. Dapprima controlliamo, interrogando la funzione **get_magic_quotes_gpg()**, se il magic quote è attivato. In caso affermativo rimuoviamo i backslash inseriti dal PHP con la funzione **stripslashes()**. Procediamo poi con mysql_real_escape_string() per effettuare un corretto escape di tutte le stringhe da inserire nella query.

Ciò che stiamo facendo è proteggere il nostro codice da uno dei più pericolosi problemi di sicurezza: l'**SQL Injection**.

Vediamo lo script completo:

```
<?php
if($_POST) {
        inserisci_record();
}
else {
        mostra_form();
}

function inserisci_record()
{
        // richiamo il file di configurazione
        require 'config.php';

        // richiamo lo script responsabile della connessione a MySQL
        require 'connect.php';

        // recupero i campi di tipo "stringa"
        $nome      = trim($_POST['nome']);
        $email     = trim($_POST['email']);
        $messaggio = trim($_POST['messaggio']);

        // verifico se devo eliminare gli slash inseriti automaticamente da PHP
        if(get_magic_quotes_gpc())
        {
                $nome      = stripslashes($nome);
                $email     = stripslashes($email);
                $messaggio = stripslashes($messaggio);
        }

        $nome      = mysql_real_escape_string($nome);
        $email     = mysql_real_escape_string($email);
        $messaggio = mysql_real_escape_string($messaggio);

        // recupero gli altri campi del form
        $sesso      = isset($_POST['sesso']) ? intval($_POST['sesso']) : 0;
        $newsletter = isset($_POST['newsletter']) ? 1 : 0;
        $attivita   = intval($_POST['attivita']);

        // verifico la presenza dei campi obbligatori
        if(!$nome)
        {
                $messaggio = urlencode("Non hai inserito il nome");
                header('location: '.$_SERVER['PHP_SELF'].'?msg='.$messaggio);
                exit;
        }

        // preparo la query
```

```
        $query = "INSERT INTO utenti
(nome,email,sesso,newsletter,attivita,messaggio)
                        VALUES
('$nome','$email',$sesso,$newsletter,$attivita,'$messaggio')";

        // invio la query
        $result = mysql_query($query);

        // controllo l'esito
        if (!$result) {
                die("Errore nella query $query: " . mysql_error());
        }

        // recupero l'id autoincrement generato da MySQL per il nuovorecord
inserito
        $id_inserito = mysql_insert_id();

        // chiudo la connessione a MySQL
        mysql_close();

        $messaggio = urlencode("Inserimento effettuato con successo
(ID=$id_inserito)");
        header('location: '.$_SERVER['PHP_SELF'].'?msg='.$messaggio);
}

function mostra_form()
{
        // mostro un eventuale messaggio
        if(isset($_GET['msg']))
                echo '<b>'.htmlentities($_GET['msg']).'</b><br /><br />';
        ?>
        <form name="form_registrazione" method="post" action="">
          <label>nome:
          <input name="nome" type="text" />
          </label>
          (obbligatorio)
          <p>
            <label>email:
            <input name="email" type="text" />
            </label>
          </p>
          <p> Sesso:
            <label>
            <input type="radio" name="sesso" value="1" />
            M</label>
            <label>
            <input type="radio" name="sesso" value="2" />
            F</label>
          </p>
          <p>
            <label>inviami newletter:
            <input name="newsletter" type="checkbox" value="1" />
            </label>
          </p>
          <p>
            <label>attivit&agrave;:
            <select name="attivita">
              <option value="0">:: seleziona ::</option>
              <option value="1">studente</option>
              <option value="2">lavoratore</option>
              <option value="3">disoccupato</option>
            </select>
            </label>
```

```
          </p>
          <p>
            <label>messaggio:<br />
            <textarea name="messaggio" cols="40" rows="5"></textarea>
            </label>
          </p>
          <p>
            <input name="invia" type="submit" value="Invia" />
          </p>
        </form>
        <?php
}
?>
```

Tutto lo script si basa su due funzioni, mostra_form() per mostrare il form all'utente e inserisci_record() per prelevare i dati inviati via POST, preparare ed eseguire la query di inserimento.

La funzione mostra_form() non necessita di particolari commenti. Si osservino le righe 72-73 che permettono di mostrare un messaggio passato via GET la cui utilità verrà spiegata in seguito.

La funzione inserisci_record() invece presenta diversi aspetti su cui vale la pena soffermarsi:

- E' buona norma eliminare eventuali spazi vuoti presenti all'estremità delle stringhe inviate dall'utente. Questi spazi, oltre a costituire uno spreco di risorse del database, possono compromettere il corretto funzionamento dei controlli sulla stringa stessa. Si pensi ad esempio alla verifica della lunghezza della stringa. Per eliminare questi spazio basta usare la funzione trim() (righe 18-20).
- Come ampiamente discusso in precedenza, è necessario effettuare l'escape dei caratteri potenzialmente "pericolosi" presenti nelle stringhe inviate dall'utente. Per prima cosa controlliamo se il PHP (probabilmente in maniera non adeguata) ha già effettuato tale operazione interrogando **get_magic_quotes_gpc()** (riga 23). In caso affermativo dobbiamo eliminare i backslash inseriti con **stripslashes()** (righe 25-27). Procediamo poi al corretto escape delle stringhe con la funzione **mysql_real_escape_string()** (righe 30-32).
- Particolare attenzione bisogna prestare al trattamento dei dati provenienti dalle checkbox e dai radio button. Se una checkbox viene selezionata dall'utente allora il suo valore sarà presente nell'array $_POST, altrimenti tale valore sarà assente. Pertanto usiamo la funzione **isset()** per effettuare tale verifica. Anche un set di radio buttons potrebbe essere privo della selezione dell'utente, per cui usiamo anche in questo caso la funzione isset().
- Se un campo della tabella ospita un intero (come i campi "sesso" e "attivita") allora è meglio assicurarci che la query contenga un valore intero da inserire. Per farlo utilizziamo la funzione **intval()**. Anche questo accorgimento serve a proteggere lo script da una SQL Injection.
- Quando si inserisce un record in una tabella è necessario assicurarsi che almeno un campo sia stato riempito. Nell'esemio riportato si è scelto di verificare che il campo "nome" non venga lasciato vuoto. Questo controllo sarebbe stato inefficace se non avessimo usato precedentemente la funzione **trim()**.
- E' importante preparare la query e conservarla in una variabile ($query) invece di passarla direttamente alla funzione mysql_query(). In caso di errore è molto utile visualizzare la query costruita con i dati dell'utente per cercare al suo interno eventuali errori causati dalla sua realizzazione dinamica.
- Una tabella contenente un campo auto_increment genera automaticamente un numero intero ad ogni inserimento di un nuovo record. Questo numero spesso rappresenta l'identificativo (ID) del record inserito e pertanto potrebbe essere utile recuperarlo dopo aver eseguito la query. La funzione **mysql_insert_id()** serve proprio ad ottenere questo numero.

Uno dei problemi più comuni in questo tipo di script è la gestione del refresh della pagina. Se l'utente, dopo aver inviato il form, ricarica la pagina (pulsante "aggiorna" del browser) è possibile che lo script di inserimento venga mandato nuovamente in esecuzione e che quindi la query di inserimento venga ripetuta. Una soluzione è quella di reindirizzare il browser verso un URL qualsiasi, anche quello dello stesso script di inserimento, in modo da perdere i dati inviati dall'utente (POST). Pertanto, eseguita l'interazione col database, lo script termina chiamando la funzione **header()** che conduce il browser all'indirizzo specificato. Conviene comunque mostrare un messaggio all'utente, per informarlo dell'esito dell'operazione. Per farlo possiamo accodare all'URL utilizzato per il redirect il messaggio da mostrare all'utente. Siccome l'URL non può contenere qualsiasi carattere allora prepariamo il messaggio convertendo gli eventuali caratteri illeciti con la funzione urlencode(). Ecco spiegato anche l'utilità della riga 73 che provvede a visualizzare il contenuto di $_GET['msg'].

Selezione dei dati e recupero dei risultati - I

di Gianni Tomasicchio

Adesso che sappiamo inserire dei dati in una tabella MySQL vediamo come poterli recuperare e mostrare all'utente. Si tratta ancora una volta di eseguire una query, in particolare bisogna lanciare una SELECT, ma ora dobbiamo occuparci anche di un altro compito, ovvero il recupero del **set di risultati** (**result set**) ottenuto.

Come visto in precedenza la funzione mysql_query() ci permette di inviare una query al server MySQL. Se ad esempio inviando una query del tipo:

```
SELECT id, nome FROM utenti
```

chiediamo al server i dati presenti nei campi "id" e "nome" della tabella "utenti". Ma dove vengono inviati questi dati? Il comportamento della funzione mysql_query() prevede che tutti i record ottenuti vengano inviati immediatamente da MySQL a PHP. Quest'ultimo li conserva nella sua memoria (in un "buffer", da cui il nome *buffered query*) ma non li rende direttamente disponibili allo script. In pratica mysql_query() non restituisce i record estratti dal database, ma ci consegna una semplice "ricevuta" che prende il nome di "MySQL result resource". Per prelevare i dati è necessaria un'ulteriore procedura detta **fetch**. Grazie alla "ricevuta" fornitaci da mysql_query() possiamo chiamare una delle funzioni di fetch e ottenere finalmente, un record alla volta, i dati appartenenti al set di risultati.

Ci sono diverse funzioni di fetch disponibili, che si distinguono per la modalità con cui restituiscono il record estratto:

- **mysql_fetch_row()** - restituisce un record del set di risultati all'interno di un array con indici numerici;
- **mysql_fetch_assoc()** - restituisce un record del set di risultati all'interno di un array associativo;
- **mysql_fetch_object()** - restituisce un record del set di risultati all'interno di un oggetto con proprietà che corrispondono ai campi del record.

Disponibile inoltre la funzione **mysql_fetch_array()** che cambia il suo comportamento in funzione del secondo parametro passatogli:

- mysql_fetch_array($ricevuta, **MYSQL_ASSOC**)
  restituisce un record del set di risultati all'interno di un array associativo. Equivalente a mysql_fetch_assoc()
- mysql_fetch_array($ricevuta, **MYSQL_NUM**)
  restituisce un record del set di risultati all'interno di un array con indici numerici - Equivalente a mysql_fetch_row()
- mysql_fetch_array($ricevuta, **MYSQL_BOTH**)
  restituisce un record del set di risultati all'interno di un array con indici numerici e associativi - Comportamento di default in caso di assenza del secondo parametro.

Se si vogliono recuperare tutti i record restituiti da una query le funzioni di fetch devono essere richiamata tante volte quanti sono i record. Queste funzioni infatti restituiscono, in formati diversi, solo un record e non tutto il set di risultati. Ad ogni nuova esecuzione, queste funzioni estraggono il record successivo, fino all'ultimo. Raggiunto l'ultimo record un'ulteriore esecuzione di queste funzioni restituisce il valore FALSE, segnalando quindi che non ci sono più dati da prelevare.

Estratti tutti i risultati non bisogna dimenticare che PHP sta conservando ancora nella sua memoria tutti i record restituiti dalla query. E' quindi possibile (e consigliato) liberare tale memoria, attraverso la funzione **mysql_free_result()**.

Tenendo presente questo comportamento delle istruzioni di fetch è facile realizzare un ciclo WHILE che ne iteri l'esecuzione fino all'estrazione completa dei record. I seguenti esempi mostrano proprio questa tecnica, applicata alle diverse funzioni di fetch.

- mysql_fetch_row

```
$ricevuta = mysql_query("SELECT id, nome FROM utenti ");

while ($row = mysql_fetch_row($ricevuta)) {
        echo 'ID: ', $row[0] , ' Nome: ', $row[1] , "\n";
}
```

- mysql_fetch_assoc

```
$ricevuta = mysql_query("SELECT id, nome FROM utenti ");

while ($row = mysql_fetch_assoc($ricevuta)) {
        echo 'ID: ', $row['id'] , ' Nome: ', $row['nome'] , "\n";
}
```

- mysql_fetch_object

```
$ricevuta = mysql_query("SELECT id, nome FROM utenti ");

while ($obj = mysql_fetch_object($ricevuta)) {
        echo 'ID: ', $obj->id , ' Nome: ', $obj->nome , "\n";
}
```

- mysql_fetch_array

```
$ricevuta = mysql_query("SELECT id, nome FROM utenti ");

while ($row = mysql_fetch_array($ricevuta, MYSQL_BOTH)) {
        echo 'ID: ', $row[0] , ' Nome: ', $row['nome'] , "\n";
}
```

Le istruzioni di fetch mostrate estraggono un record dal set dei risultati. Non è possibile specificare quale record estrarre poiché questi sono restituiti in sequenza, dal primo all'ultimo. PHP infatti tiene traccia della posizione corrente all'interno del set di risultati attraverso un puntatore, ovvero un indice che, supponendo che la query abbia prelevato dal database N record, va da 0 (il primo) a N-1 (l'ultimo).

Effettuata una query questo puntatore è inizializzato da PHP a 0 e viene incrementato di una unità ad ogni esecuzione di una funzione di fetch. Questo spiega il funzionamento dei cicli WHILE degli esempi precedenti.

Normalmente non è necessario agire su questo puntatore, poiché quasi sempre siamo interessati ad estrarre tutti i record provenienti dalla query, nell'ordine con cui sono stati prodotti ed inviati da MySQL. Se però volessimo prelevare solo alcuni record allora possiamo utilizzare la funzione **mysql_data_seek()** con cui è possibile spostarsi all'interno del set di risultati. Nel seguente esempio verrà estratto solo il primo e l'ultimo record dal set dei risultati:

```
$ricevuta = mysql_query("SELECT nome FROM utenti ORDER BY nome");

$row = mysql_fetch_assoc($ricevuta);
echo 'Primo nome: ', $row['nome'], "\n";

$numero_record = mysql_num_rows($ricevuta);
mysql_data_seek($ricevuta, $numero_record - 1);

$row = mysql_fetch_assoc($ricevuta);
echo 'Ultimo nome: ', $row['nome'], "\n";
```

Analizziamo nel dettaglio il funzionamento del precedente script:

- Per prima cosa viene lanciata la query
- Con la funzione mysql_fetch_assoc() otteniamo un record del set di risultati. Si tratta ovviamente del primo record, poiché il puntatore interno al set di risultati è inizialmente a 0
- Per sapere quanti record $numero_record sono stati estratti impieghiamo la funzione **mysql_num_rows()**
- Spostiamo il puntatore del set di risultati all'ultimo record ($numero_record - 1) con mysql_data_seek()
- Con la funzione mysql_fetch_assoc otteniamo un ulteriore record del set di risultati. Si tratta dell'ultimo record presente.

Prima di concludere con le istruzioni di fetch è doveroso citare **mysql_result()**. A differenza delle altre funzioni di fetch che prelevano un intero record dal set di risultati, mysql_result() permette di ottenere solo un campo di un particolare record (una sola cella). Ecco un esempio del suo funzionamento:

```
$ricevuta = mysql_query("SELECT id, nome FROM utenti ORDER BY id");

// mostra il nome del TERZO record restituito dalla query
echo mysql_result($ricevuta, 2,'nome');
```

Query buffered e unbuffered

Come accennato all'inizio della lezione, questo modo di effettuare una query è detto *buffered* perché è il PHP a conservare nella sua memoria tutti i record restituiti, accessibili allo script attraverso la procedura di fetch. E' possibile però lasciare a MySQL il compito di conservare i record e prelevarli, uno ad uno, direttamente con le istruzioni di fetch. Questa modalità è detta *unbuffered*. Per effettuare una query in modalità unbuffered basta usare la funzione **mysql_unbuffered_query()** al posto della classica mysql_query().

A questo punto è lecito chiedersi il motivo della presenza di due approcci distinti per la gestione del set di risultati, considerato che lo scopo ultimo rimane comunque l'estrazione dei dati dal database. Vediamo i pro ed i contro delle due tecniche:

- con le query buffered i dati vengono conservati da PHP che ha quindi la possibilità di poterli manipolare più agevolmente. E' infatti possibile utilizzare mysql_data_seek() per spostarsi liberamente all'interno dei risultati. Inoltre è possibile richiamare mysql_num_rows() per sapere subito quante righe sono state restituite dall'ultima query. mysql_data_seek() e mysql_num_rows() non sono disponibili per una unbuffered query.
- Liberando immediatamente MySQL, le query buffered permettono al server di sbloccare subito i dati coinvolti dalla query in modo che altri processi (altri utenti) possano accedervi. Se ad esempio

uno script legge un dato dal database, questo stesso dato non potrà essere modificato da un altro script fino a che non si chiude il set di risultati (mysql_free_result).

- Le query buffered costringono il PHP a conservare nella sua memoria tutto il set di risultati e questo compito può risultare oneroso se vengono gestite grosse quantità di dati.
- Le query buffered constringono lo script a sospendere la sua esecuzione fino a quando tutti i dati non hanno raggiunto il PHP, quindi il primo risultato è disponibile solo quando tutti i risultati sono estratti da MySQL, inviati e memorizzati dal PHP.
- Con una query unbuffered i risultati sono accessibili al PHP esclusivamente in maniera sequenziale ma sono disponibili non appena MySQL inizia a restituirli.
- Le query unbuffered hanno un ulteriore inconveniente. Poiché l'interazione tra PHP e MySQL non si conclude con l'invio della query ma continua per tutta la fase di fetch, per lanciare una nuova query è necessario prima prelevare tutti i risultati della precedente oppure chiudere il set di risultati (mysql_free_result).

E' chiaro quindi che non esiste una soluzione migliore rispetto all'altra. Anche se vengono privilegiate le buffered query perché permettono una migliore gestione del set di risultati, in casi di alto carico dell'applicazione e per grossi volumi di dati coinvolti, potrebbe convenire impiegare le unbuffered query.

Selezione dei dati e recupero dei risultati - II

di Gianni Tomasicchio

Con gli esempi della nella precedente lezione abbiamo visto come recuperare dei dati da un database MySQL impiegando le diverse procedure di fetch dei risultati di una query. Nella maggior parte dei casi questi dati devono poi essere mostrati all'utente, all'interno di una pagina (X)HTML, per cui è necessario prestare attenzione ad ulteriori aspetti, non meno importanti, legati sia a questioni estetiche e di rispetto degli standard, sia alla sicurezza dello script. In particolare è importante tener presente che:

- alcuni caratteri non possono essere inseriti così come si presentano all'interno di una pagina HTML ma è necessario prima convertirli nelle relative "enità HTML". Questa conversione risolve seri problemi di sicurezza che si possono verificare se tali stringhe sono state precedentemente inserite nel database da un utente.
- gli "a capo" presenti in un testo non hanno alcun effettuo sulla formattazione di una pagina HTML. Devono pertanto essere convertiti negli "a capo" dell' HTML, ovvero nei tag
 .
- in alcune circostanze, ad esempio quando si inseriscono contenuti in una tabella HTML, bisogna sostituire le stringhe vuote con qualcosa di trasparente, ad esempio l'entità
- molte informazioni memorizzate nel DB hanno una qualche forma di codifica che va correttamente trattata. Ad esempio il campo "attivita" della tabella "utenti" codifica con i numeri da 0 a 3 le attività "non dichiarata","studente", "impiegato", "disoccupato". All'utente andrà mostrata l'attività, non il numero che la rappresenta.

Vediamo quindi come tradurre in codice le precedenti osservazioni, considerando uno script che mostra i dati contenuti nella tabella "utenti" all'interno di una tabella HTML.

```
<?php
// richiamo il file di configurazione
require 'config.php';

// richiamo lo script responsabile della connessione a MySQL
require 'connect.php';

// preparo la query
$query = "SELECT nome,email,sesso,newsletter,attivita,messaggio FROM utenti";

// invio la query
$result = mysql_query($query);

// controllo l'esito
if (!$result) {
        die("Errore nella query $query: " . mysql_error());
}

$sesso_arr = array(' ','M','F');
$attivita_arr = array(' ','studente','lavoratore','disoccupato');

echo '
<table border="1">
        <tr>
                <th>Nome</th>
                <th>Email</th>
                <th>Sesso</th>
                <th>Newsletter</th>
```

```
                <th>Attivit&agrave;</th>
                <th>Messaggio</th>
        </tr>';

while ($row = mysql_fetch_assoc($result)) {
        $nome      = htmlentities($row['nome']);
        $email     = htmlentities($row['email']);
        $messaggio = htmlentities($row['messaggio']);
        $messaggio = nl2br($messaggio);

        if(!$email)     $email = ' ';
        if(!$messaggio) $messaggio = ' ';

        $sesso = $sesso_arr[$row['sesso']];
        $newsletter = $row['newsletter'] ? 'Si' : 'No';
        $attivita = $attivita_arr[$row['attivita']];

        echo "<tr>
                        <td>$nome</td>
                        <td>$email</td>
                        <td>$sesso</td>
                        <td>$newsletter</td>
                        <td>$attivita</td>
                        <td>$messaggio</td>
                </tr>";
}

echo '</table>';

// libero la memoria di PHP occupata dai record estratti con la SELECT
mysql_free_result($result);

// chiudo la connessione a MySQL
mysql_close();
?>
```

Le righe 34-36 provvedono alla conversione dei caratteri speciali con le relative entità HTML attraverso l'istruzione **htmlentities()**. La corretta visualizzazione degli "a capo" è invece affidata alla funzione **nl2br()** (riga 37)

La decodifica delle informazioni del sesso e dell'attività avviene attraverso l'uso degli array $sesso_arr e $attivita_arr. Per evitare la creazione di celle vuote abbiamo utilizzato l'entità   sia negli array $sesso_arr e $attivita_arr (righe 19-20), sia come alternativa alla stringa vuota per le variabili $email e $messaggio (righe 39-40).

Al di là delle precauzioni prese in questo script di esempio, ciò che è doveroso sottolineare è l'importanza di una corretta valutazione del formato di destinazione dei dati prelevati dal database ((X)HTML, PDF, email, file di testo, XML, ecc.). Lo script PHP infatti ha il compito di fare da tramite tra il database e l'output finale pertanto non si può limitare ad estrarre i record dalla tabella ma deve effettuare un'adeguata conversione degli stessi, operazione che se sottovalutato può condurre a seri problemi di sicurezza.

Cancellazione dei dati

di Gianni Tomasicchio

La cancellazione di record presenti in una tabella è un'operazione che richiede l'esecuzione di una semplice query. Ad esempio per cancellare dalla tabella utenti il record con id pari a 5 basta lanciare la seguente query.

```
DELETE FROM utenti WHERE id = 5
```

Per eseguire questa cancellazione in PHP è sufficiente quindi il seguente script:

```
<?php
// richiamo il file di configurazione
require 'config.php';

// richiamo lo script responsabile della connessione a MySQL
require 'connect.php';

// preparo la query
$query = "DELETE FROM utenti WHERE id = 5";

// invio la query
$result = mysql_query($query);

// controllo l'esito
if (!$result) {
        die("Errore nella query $query: " . mysql_error());
}

// chiudo la connessione a MySQL
mysql_close();

echo 'Query eseguita correttamente';
?>
```

Di solito però lo script che deve cancellare dei dati non sa a priori quale record eliminare. In un contesto interattivo è l'utente ad indicare cosa cancellare, e lo fa cliccando su di un link o selezionando in altro modo i record da eliminare. Lo script quindi dovrebbe occuparsi di mostrare i dati all'utente, permettergli di selezionare quelli da cancellare, ed infine effettuare l'operazione.

Vediamo quindi come realizzare uno script che mostra l'elenco dei record presenti nella tabella "utenti", associando a ciascuno di essi un link e una checkbox attraverso cui selezionare quelli da eliminare. Lo script verrà suddiviso in due pari, la prima adibita alla creazione dell'elenco di record disponibili, con relativi link e checkbox, la seconda per l'eliminazione del record identificato dall'id passato via $_GET o via $_POST.

```
<?php
// richiamo il file di configurazione
require 'config.php';

// richiamo lo script responsabile della connessione a MySQL
require 'connect.php';

if($_POST)
{
```

```
        $ids = isset($_POST['id']) ? $_POST['id'] : array();
        elimina_record($ids);
}
elseif(isset($_GET['id']))
{
        elimina_record(array($_GET['id']));
}
else
        mostra_lista();

function mostra_lista()
{
        // mostro un eventuale messaggio
        if(isset($_GET['msg']))
                echo '<b>'.htmlentities($_GET['msg']).'</b><br /><br />';

        // preparo la query
        $query = "SELECT id,nome FROM utenti";

        // invio la query
        $result = mysql_query($query);

        // controllo l'esito
        if (!$result) {
                die("Errore nella query $query: " . mysql_error());
        }

        echo '
        <form name="form1" method="post" action="">
        <table border="1">
                <tr>
                        <th> </th>
                        <th>Nome</th>
                        <th> </th>
                </tr>';

        while ($row = mysql_fetch_assoc($result))
        {
                $nome = htmlentities($row['nome']);

                // preparo il link per la modifica dei dati del record
                $link = $_SERVER['PHP_SELF'].'?id=' . $row['id'];

                echo "<tr>
                                <td><input name=\"id[]\" type=\"checkbox\"
value=\"$row[id]\" /></td>
                                <td>$nome</td>
                                <td><a href=\"$link\">elimina</a></td>
                        </tr>";
        }

        echo '</table>
                <br />
                <input type="submit" name="Submit" value="Elimina record
selezionati" />
                </form>';

        // libero la memoria di PHP occupata dai record estratti con la SELECT
        mysql_free_result($result);

        // chiudo la connessione a MySQL
        mysql_close();
}
```

```
function elimina_record($ids)
{
	// verifico che almeno un id sia stato selezionato
	if(count($ids) < 1)
	{
		$messaggio = urlencode("Nessun record selezionato!");
		header('location: '.$_SERVER['PHP_SELF'].'?msg='.$messaggio);
		exit;
	}

	// per precauzione converto gli ID in interi
	$ids = array_map('intval',$ids);

	// creo una lista di ID per la query
	$ids = implode(',',$ids);

	// preparo la query
	$query = "DELETE FROM utenti WHERE id IN ($ids)";

	// invio la query
	$result = mysql_query($query);

	// controllo l'esito
	if (!$result) {
		die("Errore nella query $query: " . mysql_error());
	}

	// conto il numero di record cancellati
	$num_record = mysql_affected_rows();

	// chiudo la connessione a MySQL
	mysql_close();

	$messaggio = urlencode("Numero record cancellati: $num_record");
	header('location: '.$_SERVER['PHP_SELF'].'?msg='.$messaggio);
}
?>
```

La funzione mostra_lista() si occupa di creare l'elenco dei record all'interno di un form. Per ciascuno di essi viene mostrato il nome, a cui viene applicata la funzione htmlentities() per la conversione dei caratteri nelle relative entità HTML, il link per la cancellazione diretta e la checkbox di selezione.

La funzione elimina_record() invece provvede alla cancellazione dei record selezionati. Gli id dei record vengono passati all'interno dell'array $ids (riga 10,11,15). Questi vengono prima convertiti in numeri interi per precauzione (riga 83), trasformati in una stringa di id separai da virgola (riga 86) ed infine inseriti nella query (riga 89).

Eseguita la query vengono contati i record effettivamente interessati da questa operazione attraverso la funzione **mysql_affected_rows()** (riga 100) e viene effettuato un redirect all'URL corrente per evitare possibili problemi legati al refresh della pagina, come visto nella Lezione 5. Anche in questo caso un messaggio viene accodato all'URL che verrà visualizzato all'inizio della funzione mostra_lista().

Un punto debole dello script presentato risiede nel passaggio degli id dei record da cancellare attraverso un link ($_GET) o una checkbox ($_POST), elementi comunque accessibili all'utente e pertanto soggetti a manomissione. Supponiamo ad esempio che un utente abbia i permessi

sufficienti per poter cancellare solo i record con id compreso tra 1 e 10. La funzione mostra_lista() deve quindi mostrare solo questi record ma nulla vieta all'utente di manomettere un link per la cancellazione inserendo un id superiore a 10. Senza nessun controllo quindi l'utente potrebbe eliminare record su cui non ha i permessi. Diverse le possibili soluzioni al problema. Prima di effettuare la DELETE si potrebbe verificare che l'id passato sia tra quelli accessibili all'utente impiegando la stessa logica usata da mostra_lista() per la loro individuazione. Se tale procedimento risulta oneroso mostra_lista() potrebbe salvare in sessione gli id dei record accessibili all'utente. Contro la manomissione dei link esiste anche una particolare tecnica denominata HMAC implementata nel package PEAR::Crypt_HMAC.

Aggiornamento dei dati

di Gianni Tomasicchio

L'aggiornamento dei dati presenti in una tabella MySQL è affidato ad una query di tipo UPDATE. Ad esempio, per modificare i dati associati al record con id = 5 della tabella "utenti" potremmo impiegare la seguente query:

```
UPDATE
   utenti
SET
   nome = 'Mario',
   email = 'mario@email.it',
   sesso = 1,
   newsletter = 0,
   attivita = 2,
   messaggio = 'Ciao a tutti!'
WHERE
   id = 5
```

La semplice esecuzione della query in PHP richiede la realizzazione di un banale script:

```
<?php
// richiamo il file di configurazione
require 'config.php';

// richiamo lo script responsabile della connessione a MySQL
require 'connect.php';

// preparo la query di aggiornamento
$query = "UPDATE utenti SET
                              nome = 'Mario',
                              email = 'mario@email.it',
                              sesso = 1,
                              newsletter = 0,
                              attivita = 2,
                              messaggio = 'Ciao a tutti!'
                              WHERE id = 5";

// invio la query
$result = mysql_query($query);

// controllo l'esito
if (!$result) {
        die("Errore nella query $query: " . mysql_error());
}

// chiudo la connessione a MySQL
mysql_close();

echo 'Query eseguita correttamente';
?>
```

Similmente a quanto visto nelle precedenti lezioni, anche l'aggiornamento dei dati all'interno di una reale applicazione web può richiedere ulteriori accorgimenti. Spesso infatti è necessario interagire con l'utente per determinare quale record aggiornare. Inoltre, se è l'utente a dover inserire i nuovi dati, è conveniente che possa farlo partendo da quelli attualmente presenti nel record. Un ipotetico

script interattivo per la modifica dei dati della tabella "utenti" quindi dovrà prevedere le seguenti fasi:

- mostrare un elenco dei record presenti nella tabella, associando a ciascuno un link che punta alla pagina di modifica dei relativi dati;
- mostrare un form contenente i dati correntemente associati al record selezionato;
- aggiornare i dati del record con quelli inviati dall'utente attraverso il form.

Vediamo il codice necessario a realizzare questo script:

```
<?php
// richiamo il file di configurazione
require 'config.php';

// richiamo lo script responsabile della connessione a MySQL
require 'connect.php';

if($_POST && isset($_GET['id']))
{
	aggiorna_record();
}
elseif(isset($_GET['id']))
{
	mostra_record();
}
else
	mostra_lista();

function mostra_lista()
{
	// mostro un eventuale messaggio
	if(isset($_GET['msg']))
		echo '<b>'.htmlentities($_GET['msg']).'</b><br /><br />';

	// preparo la query
	$query = "SELECT id,nome FROM utenti";

	// invio la query
	$result = mysql_query($query);

	// controllo l'esito
	if (!$result) {
		die("Errore nella query $query: " . mysql_error());
	}

	echo '
	<table border="1">
		<tr>
			<th>Nome</th>
			<th> </th>
		</tr>';

	while ($row = mysql_fetch_assoc($result))
	{
		$nome = htmlspecialchars($row['nome']);

		// preparo il link per la modifica dei dati del record
		$link = $_SERVER['PHP_SELF'] . '?id=' . $row['id'];

		echo "<tr>
```

```
                                <td>$nome</td>
                                <td><a href=\"$link\">modifica</a></td>
                        </tr>";
        }

        echo '</table>';

        // libero la memoria di PHP occupata dai record estratti con la SELECT
        mysql_free_result($result);

        // chiudo la connessione a MySQL
        mysql_close();
}

function aggiorna_record()
{
        // recupero i campi di tipo "stringa"
        $nome      = trim($_POST['nome']);
        $email     = trim($_POST['email']);
        $messaggio = trim($_POST['messaggio']);

        // verifico se devo eliminare gli slash inseriti automaticamente da PHP
        if(get_magic_quotes_gpc())
        {
                $nome      = stripslashes($nome);
                $email     = stripslashes($email);
                $messaggio = stripslashes($messaggio);
        }

        // effettuo l'escape dei caratteri speciali per inserirli all'interno
della query
        $nome      = mysql_real_escape_string($nome);
        $email     = mysql_real_escape_string($email);
        $messaggio = mysql_real_escape_string($messaggio);

        // recupero gli altri campi del form
        $sesso      = isset($_POST['sesso']) ? intval($_POST['sesso']) : 0;
        $newsletter = isset($_POST['newsletter']) ? 1 : 0;
        $attivita   = intval($_POST['attivita']);

        $id = intval($_GET['id']);

        // verifico la presenza dei campi obbligatori
        if(!$nome)
        {
                $messaggio = urlencode("Non hai inserito il nome");
                header("location: $_SERVER[PHP_SELF]?id=$id&msg=$messaggio");
                exit;
        }

        // preparo la query
        $query = "UPDATE utenti SET
                                nome = '$nome',
                                email = '$email',
                                sesso = $sesso,
                                newsletter = $newsletter,
                                attivita = $attivita,
                                messaggio = '$messaggio'
                                WHERE id = $id";

        // invio la query
        $result = mysql_query($query);
```

```
        // controllo l'esito
        if (!$result) {
                die("Errore nella query $query: " . mysql_error());
        }

        // chiudo la connessione a MySQL
        mysql_close();

        $messaggio = urlencode('Aggiornamento effettuato con successo');
        header("location: $_SERVER[PHP_SELF]?msg=$messaggio");
}

function mostra_record()
{
        // mostro un eventuale messaggio
        if(isset($_GET['msg']))
                echo '<b>'.htmlentities($_GET['msg']).'</b><br /><br />';

        $id = intval($_GET['id']);

        // preparo la query
        $query = "SELECT nome,email,sesso,newsletter,attivita,messaggio FROM
utenti WHERE id = $id";

        // invio la query
        $result = mysql_query($query);

        // controllo l'esito
        if (!$result) {
                die("Errore nella query $query: " . mysql_error());
        }

        // controllo che la SELECT abbia restituito un record
        // l'id passato via GET potrebbe essere stato manipolato
        if(mysql_num_rows($result) != 1) {
                die("l'ID passato via GET è errato");
        }

        list($nome,$email,$sesso,$newsletter,$attivita,$messaggio) =
mysql_fetch_row($result);

        $nome      = htmlspecialchars($nome);
        $email     = htmlspecialchars($email);
        $messaggio = htmlspecialchars($messaggio);

        ?>
        <form name="form_registrazione" method="post" action="">
          <label>nome:
          <input name="nome" type="text" value="<?echo $nome?>" />
          </label>
          <p>
            <label>email:
            <input name="email" type="text" value="<?echo $email?>" />
            </label>
          </p>
          <p> Sesso:
            <label>
            <input type="radio" name="sesso" value="1" <?if($sesso==1) echo
'checked="checked"'?> />
            M</label>
            <label>
            <input type="radio" name="sesso" value="2" <?if($sesso==2) echo
'checked="checked"'?>/>
```

```
            F</label>
          </p>
          <p>
            <label>inviami newletter:
            <input name="newsletter" type="checkbox" value="1" <?if($newsletter)
echo 'checked="checked"'?> />
            </label>
          </p>
          <p>
            <label>attivit&agrave;:
            <select name="attivita">
              <option value="0">:: seleziona ::</option>
              <option value="1" <?if($attivita==1) echo
'selected="selected"'?>>studente</option>
              <option value="2" <?if($attivita==2) echo
'selected="selected"'?>>lavoratore</option>
              <option value="3" <?if($attivita==3) echo
'selected="selected"'?>>disoccupato</option>
            </select>
            </label>
          </p>
          <p>
            <label>messaggio:<br />
            <textarea name="messaggio" cols="40" rows="5"><?echo
$messaggio?></textarea>
            </label>
          </p>
          <p>
            <input name="invia" type="submit" value="Invia" />
          </p>
        </form>
        <?
}
?>
```

Analizziamo nel dettaglio il funzionamento dello script. Per prima cosa è necessario capire in quale fase della procedura di aggiornamento ci troviamo. Per farlo verifichiamo la presenza di dati inviati dall'utente ($_POST) e dell'id del record passato attraverso un link ($_GET['id']). Se non è presente nessuno di questi dati allora viene richiamata la funzione mostra_lista(). Se invece è presente solo l'id allora l'utente avrà cliccato su un link e quindi mostriamo il form per la modifica dei dati, attraverso la funzione mostra_record(). Se infine è presente sia l'id, sia i dati inviati col form allora procediamo all'aggiornamento del record, richiamando aggiorna_record(). Le righe 8-17 fungono quindi da "gestore degli eventi" del nostro script

La funzione mostra_lista() si occupa di estrarre e visualizzare l'elenco dei record presenti nella tabella "utente", associando a ciascuno di essi un link per l'aggiornamento dei dati (riga 48). Le righe 22-23 mostrano un eventuale messaggio generato da aggiorna_record(), funzione discussa in seguito. Si noti come i testi dinamici inseriti nella pagina vengono adeguatamente passati prima alla funzione htmlentities() per la conversione dei caratteri in entità HTML (righe 23,45).

La funzione mostra_record() recupera i dati associati al record da modificare, individuato da $_GET['id'], e li inserisce in un form. La funzione intval() applicata a $_GET['id'] ci assicura che l'id passato alla query è costituito da un numero intero (riga 131). Effettuata l'interrogazione al database viene verificata l'effettiva restituzione del record (riga 146) come controllo sulla validità dell'id. Anche in questo caso le stringhe dinamiche da inserire nel form ($nome, $email, $messaggio) vengono preventivamente passate alla funzione htmlentities() (righe 152-154).

La funzione aggiorna_record() ha infine il compito di effettuare la query di UPDATE con i nuovi dati inviati dall'utente. Eliminati i possibili spazi vuoti alle estremità delle stringhe $nome, $email e $messaggio con la funzione trim (righe 68-70), la funzione si occupa di effettuare un corretto escape delle stesse (righe 73-83). Come ampiamente discusso nella Lezione 5, dapprima si eliminano i backslash eventualmente inseriti da PHP e poi si procede ad effettuare l'escape con la funzione mysql_real_escape_string(). Viene verificato anche se l'utente ha inserito del testo nel campo "nome" considerato obbligatorio nell'esempio. In caso contrario viene ricaricata la pagina accodando all'URL un messaggio esplicativo (righe 93-97). Se invece il controllo ha esito positivo si procede con l'esecuzione della query di aggiornamento dati e si conclude lo script con un redirect del browser che riporta l'utente alla fase iniziale. Si noti ancora una volta l'uso fatto del redirect attraverso la funzione header() (righe 96, 122) come possibile soluzione al problema del refresh della pagina, già discusso nella Lezione 6.

Tutta la logica di funzionamento dello script si basa sull'id del record passato attraverso l'URL ($_GET['id']) pertanto il codice è potenzialmente affetto da un problema di sicurezza legato alla possibile alterazione di tale valore da parte dell'utente. Continuano a valere le considerazioni fatte a riguardo nella Lezione 8.

Prima di concludere è doveroso fare un breve riepilogo ed alcune precisazioni sull'uso della funzioni mysql_num_rows() e mysql_affected_rows() incontrate più volte nei precedenti esempi:

- mysql_num_rows() restituisce il numero di record appartenenti ad un result set, prodotto ad esempio (ma non solo!) da una query di tipo SELECT;
- mysql_num_rows() non funziona correttamente con mysql_unbuffered_query(), poiché restituirà il risultato corretto solo al termine della procedura di fetch;
- mysql_affected_rows() restituisce il numero di record realmente interessato dall'ultima query di tipo INSERT, UPDATE o DELETE;
- nelle versioni di MySQL precedenti alla 4.1.2, se viene effettuata una DELETE senza la clausola WHERE, mysql_affected_rows() restituisce sempre 0;
- mysql_affected_rows() per una UPDATE non considera i record aggiornati con dati identici a quelli preesistenti. Solo i record realmente modificati vengono conteggiati. Tale comportamento può essere modificato passando il numero 2 come quinto parametro della mysql_connect(): mysql_connect("localhost", "user", "password", false, 2);
- mysql_affected_rows() per una query di tipo REPLACE restituisce la somma dei record cancellati e di quelli inseriti;
- se si è all'interno di una transazione non bisogna attendere la COMMIT per eseguire mysql_affected_rows() ma è necessario lanciarla subito dopo la query di INSERT, UPDATE o DELETE.